Navigating Hybrid Scrum Environments

Understanding the Essentials, Avoiding the Pitfalls

—

Frederik M. Fowler

Apress®

CHAPTER

6

The Scrum Development Team

The second of the three roles defined by the Scrum Framework is "developer." A Scrum developer is a technical professional who takes part in the creation of the team's product. Developers are typically organized into teams of three to nine people, and together they take responsibility for turning the Product Owner's ideas into working results ready to give to a customer.

When the Scrum Framework was described originally in 1995, the term *developer* was new to the world of software creation. Back then, there were *programmers*, *architects*, *testers*, *technical writers*, and *DBAs*. Scrum introduced the term *developer* as a generic catch-all to refer to any kind of technical professional contributing to the creation of the product.

These days (more than 20 years after the Scrum Framework was introduced), the term *software developer* has taken on a new meaning in common usage. Software developers now do the same kinds of work that "programmers" did in 1995. These days we have *developers*, *architects*, *testers*, and so on. This causes confusion and leads some organizations to believe that Scrum development teams consist of programmers only.

F. M. Fowler, *Navigating Hybrid Scrum Environments*,
https://doi.org/10.1007/978-1-4842-4164-6_6

If you recall from a previous chapter, Scrum development teams must be *cross-functional*. They must have all the different skill sets needed to deliver a finished product. In Scrum, we do not have programmers, architects, testers, and so on. We only have *developers*.

In many organizations today, there are various classifications of people with a particular skill. Someone may be a senior architect or a junior DBA. There may be lead testers and development leads as opposed to ordinary testers and developers. Development Teams may consist of people with a variety of skills, skill levels, and accumulated years of experience.

Although such differences in ability exist, the Scrum Framework recognizes no special distinctions among members of the Development Team. There are no senior developers or junior developers or lead developers. Everyone has the same title: developer.

If everyone on a team has the same title, there is an implied sense that everyone on a team is "equal" in some way. This may seem to fly in the face of common sense, because team members are not equal in their ability to contribute to the overall work product.

However, Scrum Development Team members *are* equal. They are equals in a very important way that is essential to the success of the Scrum Framework.

No two individuals can ever be equal in their capabilities, but the members of a Scrum Development Team are equal in their *responsibilities*. They are all equally responsible for the successful creation of the product.

Much of the interaction between the Development Team and the Product Owner on a Scrum Team involves negotiation. The team and the Product Owner negotiate about the work to be done in a given sprint. When agreement is reached and a scope for a sprint is defined, all the developers must be part of the agreement. Every member must feel and be equally *accountable* for the result.

This shared accountability is one of the keys to the effectiveness of the Scrum Framework. Scrum's roles are designed to make sure that accountability is aligned with *capability*. Only the Development Team has the capability to create the product. For that reason, the Development Team must have control over the creation of the product and be accountable for it.

Just as Scrum recognizes no title other than developer, it recognizes no subteams within the development team. There is no QA subteam or user interface subteam. There is only the Development Team.

The reason for this is that subteams tend to limit the scope of accountability. For example, in a traditional scenario, if a testing effort is "backed up," the DBAs might say, "That's the QA team's problem."

With Scrum, a test backup is *everyone's problem*. It belongs to the entire Development Team, because the entire Development Team told the Product Owner they would deliver something at the end of the sprint that would be "done" and ready to give to the customer.

This sense of collective accountability is very important in the Scrum Framework. Only the developers have the ability to create the product, so they must own the creation of the product. They must make all the decisions that affect the creation of the product and the way they work together.

Self-organization

Self-organization is a concept that is often misunderstood in software development organizations. It is thought, mistakenly, that self-organization is a way for teams to cut bureaucratic red tape and move decision making closer to the "front line." It is true that self-organization often leads to more efficient decision making, but this is not the reason it is a key to the success of the Scrum Framework.

Scrum Development Teams must organize themselves. They collectively must make *every* decision about what to do and how to get the work done; they must make every decision because any decision *not* made by them is a potential excuse for failing to deliver on their promises. Every time a decision is made by someone else, the developers have the opportunity to say, "Thanks for telling us what to do. We will do our best." Every decision made by someone else is owned by someone else, and the consequences are not the developers' problem.

At the beginning of each sprint, the Development Team must choose the work it will do, in agreement with the Product Owner. The team members must also make their own plan to get it done. Only then can they feel that they own the outcome of the sprint,

The Development Team must also make decisions about the tools and infrastructure it will use. Members must negotiate for the necessary server capacity, testing environments, and software development tools they wish to use. Unless they agree with the allocation of these resources to their use, they will have a ready-made excuse for every failure to deliver.

The Development Team must also make decisions about what skills the team needs to have to be cross-functional. In fact, the Development Team must have the final say as to who its team members are. Unless the team chooses its members itself, team members will have excuses to blame each other for each failure.

As problems arise, the Development Team must solve the problems itself. Members must not depend on outside help. Once again, the reason is that any help from the outside can become an excuse for poor performance ("The outsiders said they would fix it and they didn't.").

Let's take a hypothetical example. Let's imagine a development team that has a very skilled JavaScript developer and also a not-so-skilled JavaScript beginner. The skilled one is so skilled she has written books about JavaScript and teaches JavaScript classes for a local university. She has worked with JavaScript since it was invented in the 1990s. There is very little she does not know about the language.

Let's also imagine that the JavaScript beginner has very little experience. He learned JavaScript through an online training course that consisted of "ten easy lessons." He has done the online class exercises but has never written any original JavaScript code professionally. All he has done for the team so far is to make minor changes to existing code.

Let's imagine that the product the development team is creating contains a fair amount of JavaScript code. The next several features on which the team will be asked to work require a good bit of JavaScript coding. The team is not worried because they have an "expert" who can do the JavaScript work quickly and easily.

Finally, let's imagine that one day the expert gets some bad news. Her dear father, who lives 3,000 miles away, has suddenly become very ill and is in the hospital. The expert has to drop everything and rush to his bedside. As she is saying goodbye to her teammates, she lets them know that she will be unavailable for an indefinite period. It could be days, weeks, or even months.

The two questions are: Whose problem is it when the key developer becomes unavailable? Who must figure out what to do?

The answer is: The Development Team must figure it out. The Development Team owns the creation of the product. Therefore, when the unexpected happens, the Development Team must find a solution.

How can the Development Team compensate for the loss of a key resource? Here are some options:

- The inexperienced JavaScript developer can attempt to take on the work, but must be relieved of all other tasks. The Development Team can split up and share the work normally done by the JavaScript beginner to free him up for this challenge.
- If the beginner is not capable of accomplishing the work alone, the Development Team can check with other teams to see if some other team has a JavaScript expert they can "borrow" for a few sprints. They can also arrange a temporary "trade" of their beginner for the other team's expert if the other team's JavaScript needs are less challenging.

- If there is no other expert available, the Development Team can arrange for the hiring of a temporary substitute. The team would ask the Product Owner for budget money to hire a contractor and then work with the human resources department to get some candidates to interview. The team would review the candidates and make a hiring decision as a team.

In all cases just described, the entire team owns the problem of dealing with the missing JavaScript expert. Regardless of the final solution, it must be a solution owned by the entire Development Team. If it is not, and the Product Owner makes the decision, the Product Owner is solely responsible for the consequences of the decision.

For example, imagine what would happen if the Product Owner of the JavaScript-heavy product told the team, "Don't worry. I have the perfect solution. I'll hire my brother-in-law to replace the expert. He can do the job. Trust me!" The brother-in-law becomes an instant excuse for failure: "Of course we failed to deliver. We lost our expert and the Product Owner gave us his stupid brother-in-law instead. What did you expect?"

If the Development Team is to own the delivery of the product, it must have no excuses for failing to deliver. To have no excuses, the Development Team must make *all decisions about how to get the work done*. This includes all "management" decisions, including the hiring and firing of its membership, acquiring third-party resources, and securing an adequate infrastructure for doing its work.

To be a true Scrum Development Team, it must be able to solve its own problems, manage itself, and accept accountability for delivering its product.

Team Size

The members of a Scrum Development Team need to work together well. Ideally, they operate on the basis of mutual respect and trust, and are able to make decisions together quickly. When individuals on the team encounter difficulties, the others must pitch in and help. The team must succeed or fail as a whole.

When these facts are taken into consideration, it should be no surprise that team size is an important factor. If a team is too big, it may not be able to think and act as a whole.

The Scrum Guide gives a recommendation for Development Team size. It recommends that a Development Team be no smaller than three developers and no larger than nine developers. One developer is not a team; two can coordinate their activities without the need for a framework like Scrum. More than nine people can rarely agree on when to go to lunch together,

let alone how to deal with a serious challenge. The "three to nine developers" rule is based on years of practical experience on the part of Ken Schwaber and Jeff Sutherland and is a good rule of thumb.

In fact, Jeff Sutherland has made several statements in YouTube videos and at conferences in 2016 and 2017 that the ideal development team size is 4.7 people. He is quoting a recent study by the Harvard Business School as the source of this statistic.

It is important to realize, however, that the "three to nine developers" rule is a recommendation, not a requirement. The real test is whether the team can solve its own problems, manage itself, and accept accountability for creating the product.

Sometimes a team of eight or nine people can be too big. If the eight or nine people can't solve their problems, manage themselves, and accept accountability for creating the product, perhaps the thing to do is reduce the team's size.

Sometimes a team of more than nine people is not too big. I had a conversation once with a Product Owner who was distressed about the size of his team. He said, "Fred, I know my team is too big, but I hate the idea of breaking them up. They work very well together, but according to *The Scrum Guide*, the team is too big."

I asked, "Do they manage themselves?"

He answered, "Yes."

I asked, "Do they solve their own problems?"

Again, he answered, "Yes."

Finally, I asked, "Do they accept accountability for delivering the product?"

He said, "Yes!"

I said, "It seems to me the team is working well and shouldn't be split up. There's nothing broken about the team. Don't try to fix anything."

"Great!" he said happily.

I then asked him, "How many people are there on the team?"

"Twenty four," he replied.

All I could think to myself was that they must be a remarkable set of 24 people if they could indeed be self-organizing. Nevertheless, such things are possible.

Colocation vs. Geographic Distribution

The issue of colocation vs. geographic distribution of team members is a contentious one. For various reasons, some organizations create teams with members who are scattered all over the world. One of my clients has a single team whose members were in India, Europe, Central America, and California.

Is this distribution correct or incorrect? The answer lies in the basic test of whether the team is capable of self-organization. The questions to ask are

- Does the team manage itself?
- Can the team solve its problems?
- Does the team accept accountability for creating the product?

It takes a remarkable set of technicians to be able to self-organize over many different time zones. It isn't impossible, but I believe it is very rare. Also, I doubt that the developers on the distributed team I just described had any say in who they were working with. Needless to say, the amount of value they delivered each sprint was quite a bit less than other teams colocated within the company.

Cross-functionality and self-organization are the keys to aligning the accountability for creating a product with the capability of doing so. The technical people in an organization must be given authority to develop the product as they see fit. Only then can they be asked to accept responsibility for the quality and value of the outcome of their efforts.

Summary

At its heart, the Scrum Framework provides a very simple organizational model. The idea is to pair up someone who knows the business needs to be served (i.e., the Product Owner) with a team of people capable of creating the product the Product Owner needs to have created. The best results happen when the team members of technical professionals have all the tools and skills they need, and are given the freedom to decide how best to do their work by themselves.